**Министерство образования Республики Беларусь**

**Учреждение образования**

**«Белорусский государственный университет**

**информатики и радиоэлектроники»**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Кафедра интеллектуальных информационных технологий

**Отчет**

По дисциплине: Проектирование программного обеспечения в интеллектуальных системах

Лабораторная работа №1

Выполнил: Липский Р. В.,

гр. 121701

Проверил: Бутрин С. В.

**Минск 2022**

**Цель**:Изучить основные возможности языка Python

**Задание:** Реализовать на языке Python программу. Для возможности тестирования классов написать тестовую программу с меню или набор unit-тестов.

**Индивидуальное задание:** Описать класс, реализующий тип данных "Вещественная матрица". Класс должен реализовывать следующие возможности:

∙ сложение двух матриц (операторы +, +=);

∙ сложение матрицы с числом (операторы +, +=);

∙ вычитание двух матриц (операторы -, -=);

∙ вычитание из матрицы числа (операторы -, -=);

∙ произведение двух матриц (оператор \*);

∙ произведение матрицы на число (операторы \*, \*=);

∙ деление матрицы на число (операторы /, /=);

∙ возведение матрицы в степень (оператор ^, ^=);

∙ вычисление детерминанта;

∙ вычисление нормы;

**Ход выполнения:**

Реализация класса ltrx.Matrix

class Matrix:  
  
 def \_\_init\_\_(self, width=0, height=0, default\_value=None):  
 self.\_matrix: list[list[object]] = []  
 for i in range(width):  
 new = []  
 for j in range(height):  
 new.append(default\_value)  
 self.\_matrix.append(new)  
 self.\_width: int = width  
 self.\_height: int = height

Класс содержит в качестве полей \_matrix: list[list[object]], где хранятся непосредственно элементы матрицы и \_width, \_height – ширина и высота матрицы соответственно.

@property  
def size(self) -> (int, int):  
 return self.\_width, self.\_height

Size() возвращает кортеж, содержащий ширину и высоту матрицы. Имеет декоратора @property для вызова метода в виде обращения к полю класса.

def set(self, horizontal\_index: int, vertical\_index: int, value: object):  
 self.\_matrix[horizontal\_index][vertical\_index] = value  
  
def get(self, horizontal\_index: int, vertical\_index: int) -> object:  
 return self.\_matrix[horizontal\_index][vertical\_index]

Методы set и get устанавливают и возвращают элементы матрицы с указанным индексом соответсвенно.

Методы для произведения операций над матрицей выполены однотипно: если операция совершается над двумя матрицами, то это операция производится надо всеми соотвествующими элементами двух матриц. Если операция соверашется над матрицей и числом, то эта операция производится над всеми элементами и числом.

def \_add\_matrix(self, other: "Matrix") -> "Matrix":  
 self.\_check\_size\_equals(other)  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] + other.\_matrix[i][j])  
 return self  
  
def \_add\_value(self, other: object) -> "Matrix":  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] + other)  
 return self  
  
def \_sub\_matrix(self, other: "Matrix") -> "Matrix":  
 self.\_check\_size\_equals(other)  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] - other.\_matrix[i][j])  
 return self  
  
def \_sub\_value(self, other: object) -> "Matrix":  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] - other)  
 return self  
  
def \_mul\_matrix(self, other: "Matrix") -> "Matrix":  
 self.\_check\_size\_equals(other)  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] \* other.\_matrix[i][j])  
 return self  
  
def \_mul\_value(self, other: object) -> "Matrix":  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] \* other)  
 return self  
  
def \_div\_matrix(self, other: "Matrix") -> "Matrix":  
 self.\_check\_size\_equals(other)  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] / other.\_matrix[i][j])  
 return self  
  
def \_div\_value(self, other: object) -> "Matrix":  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] / other)  
 return self  
  
def \_pow\_value(self, other: "Matrix"):  
 for i in range(self.\_width):  
 for j in range(self.\_height):  
 self.set(i, j, self.\_matrix[i][j] \*\* other)  
 return self

Также для класса Matrix перегружены арифметические операторы, позволяющие производить вышеуказанные операции так же, как со встроенными типами. Реализация всех перегрузок идентична: проверяется тип второго операнда: если операнд – матрица, вызывается метод для произведения операции над двумя матрицами, иначе – над матрицей и числом:

def \_\_add\_\_(self, other):  
 if isinstance(other, Matrix):  
 return copy(self).\_add\_matrix(other)  
 return copy(self).\_add\_value(other)  
  
def \_\_iadd\_\_(self, other):  
 if isinstance(other, Matrix):  
 return self.\_add\_matrix(other)  
 return self.\_add\_value(other)  
  
def \_\_sub\_\_(self, other):  
 if isinstance(other, Matrix):  
 return copy(self).\_sub\_matrix(other)  
 return copy(self).\_sub\_value(other)  
  
def \_\_isub\_\_(self, other):  
 if isinstance(other, Matrix):  
 return self.\_sub\_matrix(other)  
 return self.\_sub\_value(other)  
  
def \_\_mul\_\_(self, other):  
 if isinstance(other, Matrix):  
 return copy(self).\_mul\_matrix(other)  
 return copy(self).\_mul\_value(other)  
  
def \_\_imul\_\_(self, other):  
 if isinstance(other, Matrix):  
 return self.\_mul\_matrix(other)  
 return self.\_mul\_value(other)  
  
def \_\_truediv\_\_(self, other):  
 if isinstance(other, Matrix):  
 return copy(self).\_div\_matrix(other)  
 return copy(self).\_div\_value(other)  
  
def \_\_itruediv\_\_(self, other):  
 if isinstance(other, Matrix):  
 return self.\_div\_matrix(other)  
 return self.\_div\_value(other)  
  
def \_\_pow\_\_(self, power, modulo=None):  
 return copy(self).\_pow\_value(power)  
  
def \_\_ipow\_\_(self, other):  
 return self.\_pow\_value(other)

Ниже приведены реализации методов Matrix::minor (нахождение минора элемента матрицы), Matrix::determinator (нахождение определителя матрицы) и Matrix::norm (нахождение нормы матрицы):

def minor(self, line: int, row: int) -> "Matrix":  
 result: "Matrix" = Matrix(self.\_width - 1, self.\_height - 1)  
 line\_skipped: int = 0  
 for i in range(self.\_width):  
 if i == line:  
 line\_skipped = 1  
 continue  
 row\_skipped: int = 0  
 for j in range(self.\_height):  
 if j == row:  
 row\_skipped = 1  
 continue  
 result.set(i - line\_skipped, j - row\_skipped, self.get(i, j))  
 return result  
  
def determinator(self):  
 if self.\_height != self.\_width or self.\_height < 1:  
 raise MatrixError("Determinator is not defined for this matrix")  
  
 if self.\_height == 2:  
 return self.get(1, 1) \* self.get(0, 0) - self.get(1, 0) \* self.get(0, 1)  
  
 result = 0  
 for i in range(self.\_width):  
 a\_minor: Matrix = self.minor(i, 0)  
 if Matrix.\_get\_sign(i, 0):  
 result += self.get(i, 0) \* a\_minor.determinator()  
 else:  
 result += -self.get(i, 0) \* a\_minor.determinator()  
 return result  
  
def norm(self):  
 result = 0  
 for i in range(self.\_height):  
 line\_sum = 0  
 for j in range(self.\_width):  
 line\_sum += self.get(j, i)  
 if result < line\_sum:  
 result = line\_sum  
 return result

Тесты для библиотеки реализованы при помощи библиотеки PyTest.

Исходный код тестов:

def test\_matrix\_equals\_must\_return\_false\_when\_matrices\_are\_not\_equal():  
 m1 = Matrix(default\_value=1, width=2, height=2)  
 m2 = Matrix(default\_value=2, width=2, height=2)  
  
 assert m1 != m2  
  
  
def test\_matrix\_equals\_must\_return\_true\_when\_matrices\_are\_equal():  
 m1 = Matrix(default\_value=1, width=2, height=2)  
 m2 = Matrix(default\_value=1, width=2, height=2)  
  
 assert m1 == m2  
  
  
def test\_matrix\_add\_must\_raise\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 + m2  
  
  
def test\_matrix\_add\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=1, width=1, height=1)  
 m2 = Matrix(default\_value=1, width=1, height=1)  
  
 assert m1 + m2 == Matrix(default\_value=2, width=1, height=1)  
  
  
def test\_matrix\_add\_must\_pass\_with\_number():  
 m1 = Matrix(default\_value=1, width=2, height=2)  
  
 assert m1 + 1 == Matrix(default\_value=2, width=2, height=2)  
  
  
def test\_matrix\_iadd\_must\_throw\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 += m2  
  
  
def test\_matrix\_iadd\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=1, width=1, height=3)  
 m2 = Matrix(default\_value=2, width=1, height=3)  
 m1 += m2  
  
 assert m1 == Matrix(default\_value=3, width=1, height=3)  
  
  
def test\_matrix\_iadd\_must\_pass\_when\_number():  
 m1 = Matrix(default\_value=1, width=1, height=3)  
 v2 = 2  
 m1 += v2  
  
 assert m1 == Matrix(default\_value=3, width=1, height=3)  
  
  
def test\_matrix\_sub\_must\_raise\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 - m2  
  
  
def test\_matrix\_sub\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=1, width=1, height=1)  
 m2 = Matrix(default\_value=1, width=1, height=1)  
  
 assert m1 - m2 == Matrix(default\_value=0, width=1, height=1)  
  
  
def test\_matrix\_sub\_must\_pass\_with\_number():  
 m1 = Matrix(default\_value=1, width=2, height=2)  
  
 assert m1 - 1 == Matrix(default\_value=0, width=2, height=2)  
  
  
def test\_matrix\_isub\_must\_throw\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 -= m2  
  
  
def test\_matrix\_isub\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=1, width=1, height=3)  
 m2 = Matrix(default\_value=2, width=1, height=3)  
 m1 -= m2  
  
 assert m1 == Matrix(default\_value=-1, width=1, height=3)  
  
  
def test\_matrix\_isub\_must\_pass\_when\_number():  
 m1 = Matrix(default\_value=1, width=1, height=3)  
 v2 = 2  
 m1 -= v2  
  
 assert m1 == Matrix(default\_value=-1, width=1, height=3)  
  
  
def test\_matrix\_mul\_must\_raise\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 \* m2  
  
  
def test\_matrix\_mul\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=1, width=1, height=1)  
 m2 = Matrix(default\_value=1, width=1, height=1)  
  
 assert m1 \* m2 == Matrix(default\_value=1, width=1, height=1)  
  
  
def test\_matrix\_mul\_must\_pass\_with\_number():  
 m1 = Matrix(default\_value=1, width=2, height=2)  
  
 assert m1 \* 1 == Matrix(default\_value=1, width=2, height=2)  
  
  
def test\_matrix\_imul\_must\_throw\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 \*= m2  
  
  
def test\_matrix\_imul\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=1, width=1, height=3)  
 m2 = Matrix(default\_value=2, width=1, height=3)  
 m1 \*= m2  
  
 assert m1 == Matrix(default\_value=2, width=1, height=3)  
  
  
def test\_matrix\_imul\_must\_pass\_when\_number():  
 m1 = Matrix(default\_value=1, width=1, height=3)  
 v2 = 2  
 m1 \*= v2  
  
 assert m1 == Matrix(default\_value=2, width=1, height=3)  
  
  
def test\_matrix\_div\_must\_raise\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 / m2  
  
  
def test\_matrix\_div\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=4, width=1, height=1)  
 m2 = Matrix(default\_value=2, width=1, height=1)  
  
 assert m1 / m2 == Matrix(default\_value=2, width=1, height=1)  
  
  
def test\_matrix\_div\_must\_pass\_with\_number():  
 m1 = Matrix(default\_value=4, width=2, height=2)  
  
 assert m1 / 2 == Matrix(default\_value=2, width=2, height=2)  
  
  
def test\_matrix\_idiv\_must\_throw\_when\_size\_not\_equals():  
 m1 = Matrix(width=1, height=3)  
 m2 = Matrix(width=4, height=2)  
  
 with pytest.raises(MatrixError):  
 m1 /= m2  
  
  
def test\_matrix\_idiv\_must\_pass\_when\_size\_equals():  
 m1 = Matrix(default\_value=4, width=1, height=3)  
 m2 = Matrix(default\_value=2, width=1, height=3)  
 m1 /= m2  
  
 assert m1 == Matrix(default\_value=2, width=1, height=3)  
  
  
def test\_matrix\_idiv\_must\_pass\_when\_number():  
 m1 = Matrix(default\_value=4, width=1, height=3)  
 v2 = 2  
 m1 /= v2  
  
 assert m1 == Matrix(default\_value=2, width=1, height=3)  
  
  
def test\_matrix\_pow\_must\_pass\_with\_number():  
 m1 = Matrix(default\_value=2, width=2, height=2)  
  
 assert m1 \*\* 3 == Matrix(default\_value=8, width=2, height=2)  
  
  
def test\_matrix\_ipow\_must\_pass\_when\_number():  
 m1 = Matrix(default\_value=2, width=1, height=3)  
 v2 = 3  
 m1 \*\*= v2  
  
 assert m1 == Matrix(default\_value=8, width=1, height=3)  
  
  
def test\_matrix\_minor():  
 m1 = Matrix(default\_value=0, width=3, height=3)  
 expected = Matrix(default\_value=0, width=2, height=2)  
  
 assert expected == m1.minor(0, 0)  
  
  
def test\_matrix\_determinator\_must\_be\_zero\_when\_all\_equal():  
 m1 = Matrix(default\_value=0, width=3, height=3)  
  
 assert 0 == m1.determinator()  
  
  
def test\_matrix\_determinator\_must\_pass():  
 m1 = Matrix(default\_value=2, width=4, height=4)  
 m1.set(0, 0, 9)  
 m1.set(1, 1, 9)  
 m1.set(2, 2, 9)  
 m1.set(3, 3, 9)  
  
 assert m1.determinator() == 5145  
  
  
def test\_matrix\_norm\_must\_pass():  
 m1 = Matrix(default\_value=2, width=3, height=3)  
 m1.set(0, 0, 1)  
 m1.set(0, 1, 2)  
 m1.set(0, 2, 3)  
  
 assert m1.norm() == 7

Список использованных источников:

1. Python 3.11.2 Documentation - <https://docs.python.org/3/>

2. PyTest Full Docs <https://docs.pytest.org/en/7.1.x/contents.html>